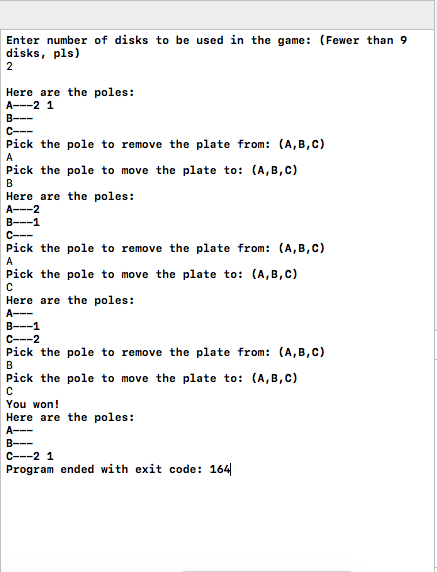
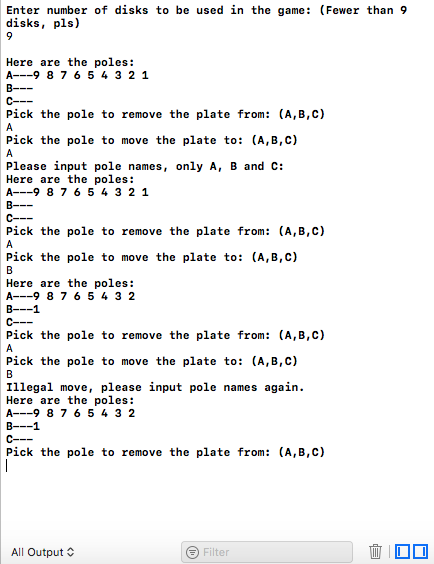
Task 1: None

Task 2: Include in the lab report a screenshot(s) of the output of your tests.

1. Here’s a screenshot of the entire game



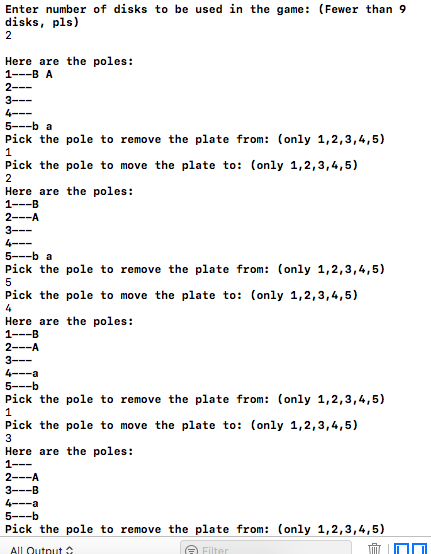
1. Here are some illegal moves: Moving one a bigger plate onto a smaller one or Moving a plate to the same spot.

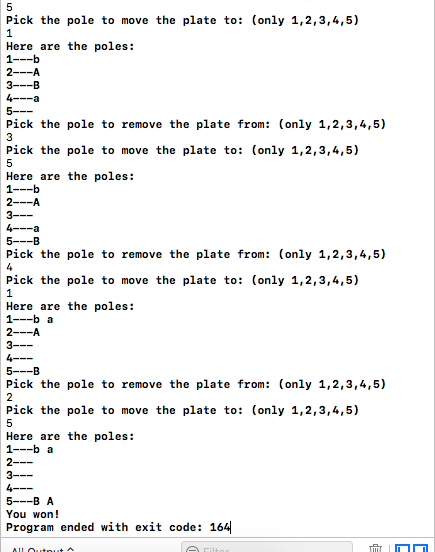


\* The reason we have 9 plates maximum is there are 9 digits

Task 3: Include in the lab report a screenshot(s) of the output of a test. Include a discussion of the different strategies you tried (cooperative, defensive, ignoring the other player completely, etc…).

1. Here are some screenshots of the game being played:



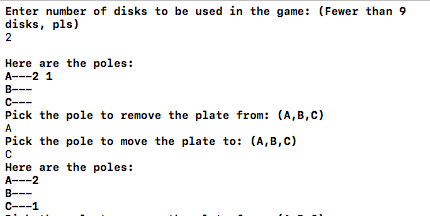


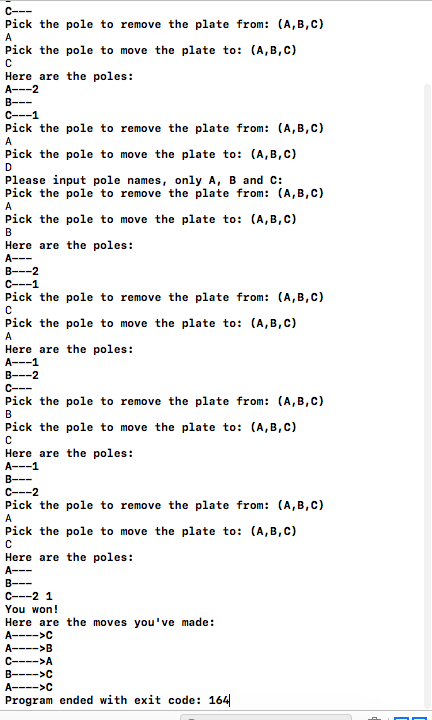
From what I understand, it’s a cooperative game, where the game is won when both players achieves their purposes. If it was competitive, once player 1 move 1 disk, player 2 can just move that disk back to the original position. If a player try to block another block of the other play, player2 can just remove that block. Thus it wouldn’t make an interesting game being competitive. More like a race.

If it’s a cooperative game, since there’s an optimal way to win the game as fast as possible, both players must understand that and commit to the greater good everymove, which means being aware and support the other player’s every move, making it as easy as possible so both wins.

Task 4: Include in the lab report a screenshot(s) of the output of a test. Include a discussion of how you designed your move data structure and how you had to modify the code for automating the game.

1. Here are the screenshots to the code:





\* As you can see, I input the wrong pole name “D” and I designed so the list of moves wouldn’t include that move.

My move data structure is simple, only including the names of the pole to move from and the pole to move to. The class queue is much more complicated. Since we’re using a vector, a pointer is not needed. Since the index is built into the STL vector, we don’t need to use rear and front index. Since the vector when initialized will fill all uninitialized spot with 0, I didn’t initiated it with a default size at all, and only push\_back() continuously. Therefore a size private variable is not necessary. I created a function that returns the size of the vector.

Gameplay: We asked the player to input 2 characters, to indicate which pole to move from and to. We created a while loop, and it keeps looping until the winning condition is met. We put the 2 inputs into the move struct, then push that struct into the list of moves. If they input the error inputs, the ifs escapes and goes back to the beginning of the loop.